# Solving Double Pendulum Balance with Policy Gradients

REUBEN DUBESTER, Oberlin College

The double pendulum balance problem is a classic problem in machine learning, and has been solved using a wide variety of methods. The specific version of the problem that I attempted to solve was the Acro-Bot environment provided by the OpenAI project. Solving this problem involved overcoming many of the typical machine learning problems, such as anticipating future rewards, handling large state-actions spaces, and dealing with locally optimal solutions. I chose to use policy gradients in my attempt to solve the double pendulum problem, and chose Google's Tensorflow library to implement this approach. I tested multiple reward functions, model parameters, and action selection strategies to find the optimal learning conditions. I also compared the performance of this technique to more basic strategies, and how it performed on easier problems. Overall, the learning agent was able to obtain substantial increases in reward over time, although it obtained this improvement using undesirable strategies.

CCS Concepts: • **Computing methodologies** → Reinforcement learning; Neural networks; Markov decision processes;

Additional Key Words and Phrases: Policy Gradients

**ACM Reference Format:**
Reuben Dubester. 2017. Solving Double Pendulum Balance with Policy Gradients. 0, 0, Article 0 (December 2017), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

When starting this project, I had no substantial knowledge of reinforcement learning, neither in theory nor in implementation. I began by searching for articles and tutorials online, but was frustrated by a lack of good entry points into the field; every article was either overly simplified to the point of being irrelevant, or incredibly complex and aimed at other experts in the field. Additionally, both the Tensorflow and OpenAI platforms were daunting to dive into, and it was a challenge to find helpful resources. I started with the very basics, using random search to solve CartPole, the most basic Classic Control problem of the OpenAI gym environments. From that point, I experimented briefly with some more complex approaches, such as genetic algorithms and Q-Learning, before deciding on policy gradient learning as the focus of my project. I also chose to tackle the more challenging Acrobot gym environment, as I felt it was a reasonable step up in complexity from CartPole, without being overly daunting. Although in the end I achieved reasonable results with this method and vastly improved my understanding of reinforcement learning, there is still a substantial amount of progress to be made, and much more to learn in the future.

This report contains a descriptions of the CartPole and Acrobot environments, explanations of the learning techniques I used to solve them, and the results obtained from applying these techniques. In doing this project, I encountered a lot of the classic problems and pitfalls associated with designing learning agents, and this report includes discussions of those problems, their significance to the broader field of machine learning, and how I attempted to overcome them. I'll also discuss potential future approaches to refine the solutions presented, and overall lessons learned.

Author's address: Reuben Dubester, Oberlin College, rdubeste@oberlin.edu.



## 2 PROBLEM

### 2.1 Cartpole

CartPole is the most basic Classic Control environment on the OpenAI platform, and has been solved successfully with a wide variety of techniques in the past. As the name implies, CartPole consists of a cart which can be moved left or right along a track by applying small amounts of force, and a pole which starts balanced upright on the cart, but is free to rotate in either direction. The task of the environment is to keep the pole balanced on the cart for as long as possible, without the cart moving too far in either direction. Because of the instability of the inverted pendulum, this requires constant adjustment from the cart. The state information data available to the learner consists of the position of the cart, velocity of the cart, angle of the pole, and angular velocity of the pole. This environment is complex, but has certain features which make it drastically easier to solve compared to other environments, which will be discussed later in more detail.

### 2.2 Acrobot

Acrobot is another one of the OpenAI Class Control environments. It consists of a double pendulum that begins the simulation hanging straight downwards. The goal of this environment is to swing the tail end of the second linked pendulum up to a height of a pendulums length above the pivot point, and it can do this by applying a small rightward, or leftward force to the pivot of first pendulum at each time step. The amount of force that can be applied in a single step is small enough that just applying the same directional force every time step is not enough to rotate the pendulum more than a few degrees, so successfully swinging the double pendulum up to height necessitates building up momentum at the bottom of the swing. This presents a significant long term planning problem, that makes the environment substantially more complex than CartPole. The state information consists of the angles and angular velocities of the two pendulums.

## 3 SOLUTION

### 3.1 Random Search

Before I tried implementing any real reinforcement learning algorithms, I wanted to get a baseline to compare further results to, as well as increase my understanding of the API for the OpenAI platform. To do this, I solved the CartPole environment using a random search strategy. I created a simple linear model, which takes in the

state of the world at each time step, multiplies these values with a set of weights, and then makes a decision based on this output. This results in a new state and a reward. For CartPole, this reward is always 1 as long as the pole is upright. After a set amount of time steps, the environment is reset and the weights are randomized. This is repeated until a set of weights are found which successfully solve the problem. In the case of CartPole, solving the problem is defined as reaching a cumulative reward of 200. I also tried this approach on the Acrobot environment. The linear model was unchanged, but the reward function was defined to be 1 if the end of the double pendulum is above a height of 1 (or one pendulum length above the pivot) and -1 otherwise. Additionally, there was no set criteria for the problem to be considered solved. Instead, the simulation ran for a set number of time steps, logged the cumulative reward gained over the course of that episode, then reset. All the math for this approach was implemented using the Numpy library, and the training procedure was implemented using my own code.

### 3.2 Policy Gradients

After solving CartPole, I turned my attention to policy gradients. This is a much more complex strategy for learning, and one that I hoped would be able to handle many of the challenges associated with solving the Acrobot environment. The basic training loop of the agent is as follows. For a given amount of time steps, the agent takes in the state of the world and chooses and performs an action.In order to translate states into actions, a neural network is used. The neural network I used has an input layer that takes in the state information, two fully connected hidden layers, and an output layer with one neuron for each possible action. The output layer applies a softmax function over the outputs, and the value of each output neuron is represents the models assessment of the probability that that action is the correct one to take given this state. The action can then be chosen probabilistically based on these values, or simply by taking the action with the highest probability. Once an action is performed, the agent then records in an array the state, the action chosen, the reward obtained from that action, and the new state. Once the training episode is concluded, that array is fed into the optimizer. Before being fed in however, the rewards are put through a discount function. The discount function basically allows each state-action pair to not only be responsible for the immediate reward received, but also partly responsible for the future rewards gained from actions taken later in the episode. This will allow the model to more effectively plan for the future.

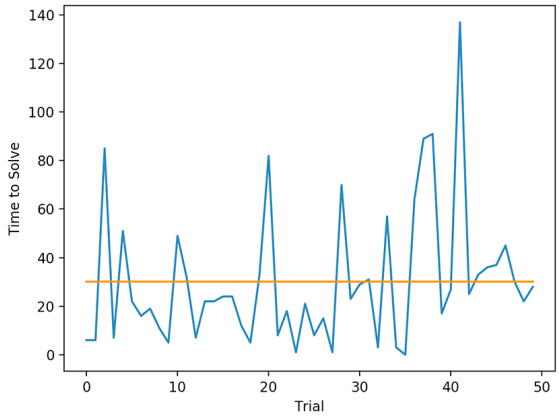
After the reward function is processed, the array of data generated during the training episode is fed into the learner. The optimizer calculates the gradients for the model. These gradients reflect how the loss function changes as the values of the weights within the neural network change. The loss function is the negated value of the mean of the rewards obtained from a given action times the probability that that action was chosen. The basic reasoning for this loss function is that the goal of the agent in any given state is to maximize the probability of choosing the correct action, which is the action that yields the highest reward. Finally, the optimizer uses gradient descent to nudge the values of the weights within the neural network towards values that will minimize the loss. After the values have been adjusted, a new training episode begins using the updated neural network, and the training loop repeats. The optimizer I am using is the Adam Optimizer. This, the neural network, and the gradient calculation are provided by the Tensorflow library.

## 4 PERFORMANCE EVALUATION

I evaluated the performance of the agent by examining the change in reward over multiple training episodes and by analyzing the ultimate behavior of the model. More sophisticated analysis was not really necessary for this project, as just by looking at the cumulative reward values it was fairly obvious when the performance of the learner was increasing versus when it was stuck. I experimented by tweaking the learning rates and changing the architecture of the neural network, but found that these changes made very little overall difference in performance.

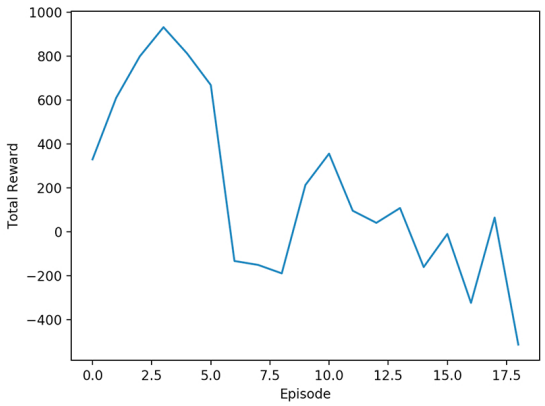
## 5 RESULTS

For CartPole, I found that random search was very effective at solving the environment. Over 50 trials, I found that on average random search found the correct model parameters to achieve a reward of over 450 after only 30 episodes, as is shown in the graph below.




I believe that random search is so effective on this task due to two factors. First of all, the task is not that complex. There are only 4 attributes for each state, thus only 4 parameters to tune. Second, the environment is very forgiving; There are a wide range of possible strategies that will lead to success, and the agent does not need to be very precise when acting to keep the pole up in the air. This means that instead of having to learn strategies that might with I think the lesson to be learned from this result is that sophisticated learning strategies are often overkill, and that it is worth experimenting with simple strategies like random search before spending a substantial amount of time working towards a more complicated solution. Because of the high effectiveness of random search I decided not to test the policy gradient approach with CartPole.

For AcroBot. I found that random search was not an effective strategy. The learner made very little progress, and even after large amounts of trials the end behaviour of the learner did not gain substantial reward. On the other hand policy gradients were effective at increasing the reward obtained by the agent over time, however they did so in a way that was not desirable. Shown below is the reward gained by the learner over 500 episodes. Although the growth in reward is not very stable, and the learner often backtracks considerably, the overall trend is very positive.




I had hoped that the agent would learn swing up the double pendulum, then balance it on its end. I envisioned this as the optimal behavior of the learner, because holding the end of the pendulum in the air for as long as possible maximizes the total time that the end of the pendulum is at maximum height. While the learner did learn to build up speed and swing the pendulum up to height, instead of balancing the pendulum upright, in almost all cases the learner simply began swinging the pendulum in a circle as fast as possible. I believe there are several reasons behind this behavior. First, although spinning around in circles is not a completely optimal behavior, it is better than just hanging idly or swinging back and forth at the bottom of the arc. When the pendulum is at the highest point in its rotation, it is gaining reward. Second, the learner is at risk of ending up in a self-reinforcing cycle. Once the pendulum has spun up to high speed, slowing down the rotation to get the pendulum back under control will actually reduce the reward obtained by the agent, as in the process of slowing down the rotation the pendulum will complete fewer rotations and thus spend less time upright. Another aspect of the model that makes the cycle self-reinforcing is that once the pendulum is at a high speed, there is no real way that it can learn to break out of that behavior. That is because even if the model were to randomly choose the correct action, which is slowing down the pendulum, because of the low force exerted and the high velocity of the pendulum that decision will not substantially alter the overall behavior of the system, so the agent won't learn that that action was actually correct. To get around this behavioral problem, I experimented with altering the reward function. I included a penalty for going over a certain speed when the pendulum is above a given height, and I added a bonus reward for having a low speed when the end of the pendulum is in the air. These improvements did help the pendulum adopt slightly better behavior, but were not totally effective at mitigated the spinning.

## 6 CONCLUSIONS AND FUTURE WORK

What most struck me about the results of this project was the behavior of the Acrobot learner, and the way that it learned to optimize reward in a way that was very unexpected to me and vastly different than what I would have hoped the end behavior would be. Although initially humorous, my inability to eradicate this behvior eventually grew into a source of considerable frustration, and suggests that the learned behavior is indicative of a deep flaw in the method I have chosen to approach this problem with. This suggests broader implications for the future of machine learning. When a learner is left to find a solution on its own, it can often find solutions and behaviors that are unpredictable and undesirable. As the power of machine learning and the influence of algorithmic decision making grows stronger and stronger in our every day lives, and more serious problems are tackled with computer learning, it is important to be remember that our assessment of an optimal solution can be very different than what may be produced when the system is left unchecked. The partial solution I found in this case of refining the reward function based on human knowledge is a reminder of the value of human input and judgment when designing systems that handle complex tasks. In the future, I would like to extend this problem by continuing to experiment with different reward functions, and attempt to implement other reinforcement learning methods to compare with policy gradients. Although the policy gradient approach gave me decent results, I believe that other reinforcement learning methods that are more adept at long term planning may be more effective at avoiding the negative behavior produced through my initial approach.

## ACKNOWLEDGMENTS

Thanks to Professor Adam Eck for teaching a fantastic machine learning class.